AMENDMENTS TO THE SPECIFICATION:

Replace the title of the invention on page 1 with the following:

WRITEBACK POLICY FOR MEMORY BASED ON STACK TREND INFORMATION

Replace paragraph **[0001]** with the following:

[0001] This application claims priority to U.S. Provisional Application Serial No. 60/400,391 titled "JSM Protection," filed Jul. 31, 2002, incorporated herein by reference. This application also claims priority to EPO Application No. 03291914.4, filed Jul. 30, 2003 and entitled "Write Back Policy For Memory," incorporated herein by reference. This application also may contain subject matter that may relate to the following commonly assigned co-pending applications incorporated herein by reference: "System And Method To Automatically Stack And Unstack Java Local Variables," Serial No. 10/632,228[[___]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35422 (1962-05401)~~; "Memory Management Of Local Variables," Serial No. 10/632,067[[___]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35423 (1962-05402)~~; "Memory Management Of Local Variables Upon A Change Of Context", Serial No. 10/632,076 [[___]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35424 (1962-05403)~~; "A Processor With A Split Stack," Serial No. 10/632,079[[___]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35425(1962-05404)~~; "Using IMPDEP2 For System Commands Related To Java Accelerator Hardware," Ser. No. 10/632,069[[___]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35426 (1962-05405)~~; "Test With Immediate And Skip Processor Instruction," Ser. No. 10/632,214[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35427 (1962-05406)~~; "Test With Immediate And Skip Processor Instruction," Ser. No. 10/632,084 [[____]],

filed Jul. 31, 2003~~, Attorney Docket No. TI-35248 (1962-05407)~~; "Test And Skip Processor Instruction Having At Least One Register Operand," Ser. No. 10/632,084 [[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35248 (1962-05407)~~; "Synchronizing Stack Storage," Ser. No. 10/631,422[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35429 (1962-05408)~~; "Methods And Apparatuses For Managing Memory," Ser. No. 10/631,252[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35430 (1962-05409)~~; "Methods And Apparatuses For Managing Memory," Ser. No. 10/631,205[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35432 (1962-05411)~~; "Mixed Stack-Based RISC Processor," Ser. No. 10/631,308[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35433 (1962-05412)~~; "Processor That Accommodates Multiple Instruction Sets And Multiple Decode Modes," Ser. No. 10/631,246[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35434 (1962-05413)~~; "System To Dispatch Several Instructions On Available Hardware Resources," Ser. No. 10/631,585[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35444 (1962-05414)~~; "Micro-Sequence Execution In A Processor," Ser. No. 10/632,216[[____]], filed July 31, 2003)~~, Attorney Docket No. TI-35445 (1962-05415)~~; "Program Counter Adjustment Based On The Detection Of An Instruction Prefix," Ser. No. 10/632,222[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35452 (1962-05416)~~; "Reformat Logic To Translate Between A Virtual Address And A Compressed Physical Address," Ser. No. 10/632,215[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35460 (1962-05417)~~; "Synchronization Of Processor States," Ser. No. 10/632,024[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35461 (1962-05418)~~; "Conditional Garbage Based On Monitoring To Improve Real Time Performance," Ser. No. 10/631,195[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35485 (1962-05419)~~; "Inter-Processor Control," Ser. No. 10/631,120[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35486 (1962-05420)~~; "Cache Coherency In A Multi-Processor System," Ser. No. 10/632,229[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35637 (1962-05421)~~; "Concurrent Task Execution In A Multi-Processor, Single Operating System Environment," Ser. No. 10/632,077[[____]], filed Jul. 31, 2003~~,~~

~~Attorney Docket No. TI-35638 (1962-05422)~~; and "A Multi-Processor Computing System Having A Java Stack Machine And A RISC-Based Processor," Ser. No. 10/631,939[[____]], filed Jul. 31, 2003~~, Attorney Docket No. TI-35710 (1962-05423)~~.

Replace paragraph **[0029]** with the following:

Replace paragraph **[0019]** with the following:

[0019] Controller 26 includes compare logic 42 and word select logic 44. The controller 26 may receive an address request 45 from the AGU 22 via an address bus, and data may be transferred between the controller 26 and the ALU 24 via a data bus. The size of address request 45 may vary depending on the architecture of the system 10. Address request 45 may include an upper portion ADDR[H] that indicates which cache line the desired data is located in, and a lower portion ADDR[L] that indicates the desired word within the cache line. Although FIG. 2 depicts a fully associative cache 14, this configuration is portrayed solely for the sake of clarity. It should be noted that the subject matter disclosed herein equally applies to any order of multi-way set associative cache structures where the address 45 is split into three parts--an upper portion ADDR[H], a middle portion ADDR[M], and a lower portion ADDR[L]. Compare logic 42 may compare the requested data address to the contents of the tag memory 36. If the requested data address is located in the tag memory 36 and the valid bit 38 associated with the requested data address is enabled, then the cache line may be provided to the word select logic 44. Word select logic 44 may determine the desired word from within the cache line based on the lower portion of the data address ADDR[L], and the requested data word may be provided to the processor 12 via the data bus. Dirty bits 39 may be used in conjunction with the write back policy. On a write request, if the data is modified within the cache without being modified in main memory, the corresponding dirty bit associated with the cache line where the data resides is set to indicate that the data is not coherent with its value in main memory. Subsequently, dirty cache lines may be evicted from cache space as space is needed. Dirty lines that are evicted during a line replacement caused by a cache miss need to be written back to the main memory,

whereas non-dirty lines may be simply overwritten. Consequently, in cache memory structures having write back policies, data are made coherent within the main memory when they are evicted from the cache either when selected as victim line by the replacement policy (e.g. LRU, random), or when an explicit request like a "clean cache" command occurs. The write back policy may be adapted for certain data types in order to improve overall system performance. Decode logic 20 processes the address of the data request and may provide the controller 26 with additional information about the address request. For example, the decode logic 20 may indicate that the requested data address belongs to the stack 32 (illustrated in FIG. 1). Using this information, the controller 26 may implement cache management policies that are optimized for stack based operations as described below.

Replace paragraph **[0021]** with the following:

[0021] In accordance with some embodiments, data may be written to cache memory 14 on a cache miss without allocating or fetching cache lines from main memory 16, as indicated in co-pending application entitled "Methods And Apparatuses For Managing Memory," filed Jul. 31, 2003, Ser. No. 10/631,185 ~~Jul. 31, 2003 (Atty. Docket No.: TI-35430)~~. FIG. 3 illustrates a system optimization in the cache management 48 related to a write back policy for stack data that may be implemented by the controller 26. Block 50 illustrates a write request for stack data. As a result of the stack data write request, the AGU 22 may provide the address request 45 to the controller 26. Controller 26 then may determine whether the data is present in cache memory 14, as indicated by block 52. If the data is not present within cache memory 14, a "cache miss" may be generated, and cache memory 14 may allocate a new line per block 54. If the data is present within the cache, a "cache hit" may be generated, the data may be updated, and the dirty bit of the corresponding line may be set as indicated in block 56. During the cache update, the

cache controller determines if the address corresponds to the last word of a cache line per block 58. If the address does not correspond to the last word in the cache line, the cache is updated per block 56. Otherwise, if the address corresponds to the last word, the cache controller 26 writes back the line to the main memory as indicated in block 60. When the line is written back to main memory, the dirty bit is cleared indicating that cache and main memory hold coherent data. This creates potentially additional data transfer compared to a standard write back policy but provides the advantage of reducing the latency when a coherent view of the stack at a lower level of memory is required.

Replace paragraph **[0023]** with the following:

[0023] Referring to FIG. 4, an instruction pipe 66 of processor 12 is illustrated including instructions INST.sub.0 through INST.sub.N. In a processor there may be several stages between the execution of a data memory access and the decode of an instruction. In addition, these stages might include some predecoding stages, such that the information supplied to the trend logic 21 might come from some decode and predecode logic. In the preferred embodiment, the memory access stage and the decode stage are separated by several pipe stages. Accordingly, the trend logic 21 may use one or more signals supplied by the decoder logic 20. As instructions progress within the pipe their effect on the stack 32 is taken into account by the trend logic to generate the appropriate trend information to the cache controller. For example, trend logic 21 may receive CURRENT 68 and FUTURE 70 to generate the trend information about the stack 32. When the instruction preceding instruction INST.sub.0 is accessing memory, the trend logic 21 receives information from decode logic 20 and sends to the cache controller CURRENT 68 stack trend information relating to what effect instruction INST.sub.0 will have on the stack 32. In addition, the trend logic may send information indicating FUTURE 70 stack trend information corresponding to the following instruction INST.sub.1 through

INST.sub.N. For example, if INST.sub.0 is an "iload" instruction (which pushes an operand on to the stack 32), then this may cause the stack 32 to increase by 1, whereas if INST.sub.0 is an "iadd" instruction (which pops two operands from the stack adds them together and pushes the result back on the stack), then this may cause the stack 32 to decrease by 1. In a similar fashion, FUTURE 70 represents a signal coming from the decoder 20 and potentially a predecode logic corresponding to a predetermined number of instructions following INST.sub.0 within the instruction pipe 66 (e.g., INST.sub.1 through INST.sub.N), and providing information on the evolution of the stack for the subsequent instructions to trend logic 21. Trend logic 21 may then utilize the CURRENT 68 and FUTURE 70 stack trend information to determine a net stack trend. For example, if INST.sub.1 increases the stack by 1, INST.sub.2 decreases the stack by 1, and INST.sub.3 decreases the stack by 2, then the net stack trend is to be decreased by 2. Using the net stack trend, the trend logic 21 may maintain more than one dirty cache line in cache memory 14 where the trend information may indicate that some of the data in the dirty cache lines are going to be consumed by the subsequent instructions.

Replace paragraph **[0026]** with the following:

[0026] As was described above, stack based operations, such as pushing and popping data, may result in cache misses. The micro-stack 25 may initiate the data stack transfer between system 10 and the cache memory 14 that have been described above as write access on stack data. For example, in the event of an overflow or underflow operation, as is described in copending application entitled "A Processor with a Split Stack," filed Jul. 31, 2003, Ser. No. 10/632,079 [[__]] ~~(Atty. Docket No.: TI-35425)~~ and incorporated herein by reference, the micro-stack 25 may push and pop data from the stack 32.